信用规则与信用评估

学校：同济大学

参赛队员信息：

姓名	专业
操瑞行	统计学
林镜鸿	数学与应用数学
吴博	数学与应用数学
李帅	计算机科学与技术
杜旭岩	数理金融

指导老师：钱志坚

目录

摘要

信用评分体系对于企业以及金融机构有着重要的意义。在本文中我们基于某贷款机构的历史业务数据，建立信用评分模型。

我们先进行数据预处理，数据可视化，初步观察出数据的大体规律。然后进行数据清洗，把赶脏数据处理，对于异常值合理舍弃，缺失值处理的方法，包括如下几种。（1） 直接删除含有缺失值的样本；（2） 根据样本之间的相似性填补缺失值；（3） 根据变量之间的相关关系填补缺失值。在本案例中，对于 AGENT，由第二步的数据可视化发现与 Y 值关系程度很低，并且缺失值太多，考虑舍弃。随后对于 salary 用随机森林算法进行填补。随机森林算法对缺失值进行合适的处理，使得数据集具有完整性。特征工程阶段，为选取信用评分的合适指标，我们先将所有相关表格以 ID 为主键，左连接，合成新的完整表格，再用 LassoCV 算法进行初步特征选择。进一步，采用分箱处理,WOE 分析，以及相关性分析与 IV 筛选的方法筛选出更为精准的特征，并使用 WOE 转换将其转换为 WOE 值，以便于模型的建立。

然后，我们用 train_test_split 方法来随机切分已生成的 WOE 数据集，并将其 1/4 作为测试集，3/4 作为训练集。采取 Logistic Regression，Decision Tree Classifier,Ridge Classifier,SGD Classifier 四种模型，并用上述得出的训练集带入模型对象进行训练，得出适合本案例的模型，并比较四种模型的正确率，得出最佳模型是决策树模型。最后，再用 bagging 方法进行模型融合，进一步有效缓解数据集过小带来的过拟合问题，提高决策树模型的正确率到 96.7%。将 test 集带入训练出的决策树模型，可以进一步预测出 test 的集的 Y 值为 0 的概率以及为 1 的概率，即为违约率，从而建立出完善的信用评分模型。

表格和插图清单

一、问题描述

当下互联网金融蓬勃发展，呈现出多种多样的业务模式和运行机制。金融机构能够突破时间和地域的约束，在互联网上为有融资需求的客户提供更快捷的金融服务。通过互联网技术，加快业务处理速度，带给用户更好的服务体验。但同时存在着信用风险和用户欺诈等问题，急需通过信用评分模型提高风险控制水平。

征信机构利用采集到的丰富信息对个人进行综合信用评价。在丰富海量的个人信用历史和信用行为数据基础上，采用数据挖掘方法得出的信用行为模式能够更加准确地预测个人未来的信用表现，能够提高操作的效率，降低授信成本，精确估计消费信贷的风险，是金融机构内部评分不可替代的重要工具。因此，建立精准的信用评分体系对于企业以及金融机构有着重要的意义。

二、数据描述

（一）数据预处理

1、数据清洗

在对训练集数据处理之前，需要对训练集数据的缺失值和异常值情况进行了解。Python 内有 describe()函数，可以了解数据集的缺失值、均值和中位数等。

结果显示如表 1 和表 2 所示：

表 1

REPORT_ID	30000	non-null	int64
ID_CARD	30000	non-null	object
LOAN_DATE	30000	non-null	object
AGENT	8952	non-null	object
IS_LOCAL	30000	non-null	object
WORK_PROVINCE	27742	non-null	float64
EDU_LEVEL	26942	non-null	object
MARRY_STATUS	30000	non-null	object
SALARY	8864	non-null	float64
HAS_FUND	29998	non-null	float64
Y	30000	non-null	int64

表 2

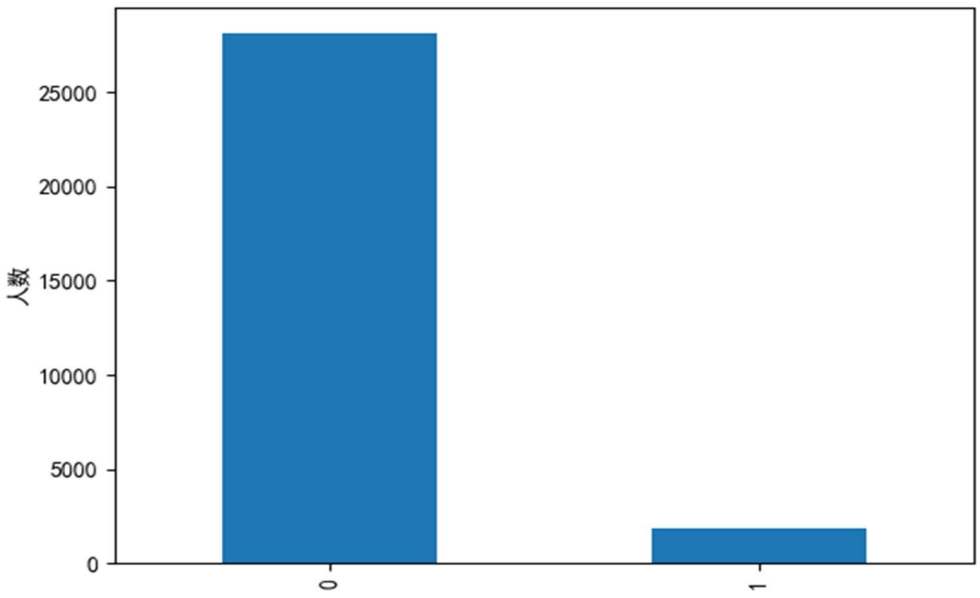
	REPORT_ID	WORK_PROVINCE	SALARY	HAS_FUND	Y
count	3.00E+04	27742	8864	29998	30000
mean	3.26E+06	308176.4141	3.618682	0.455364	0.0625
std	1.56E+06	101940.8799	1.415137	0.498012	0.242065
min	8.79E+03	110000	1	0	0
25%	2.02E+06	230000	3	0	0
50%	3.44E+06	330100	3	0	0
75%	4.63E+06	370000	4	1	0
max	5.63E+06	650000	7	1	1

由上表可知，train 集一共 30000 行数据，其中 AGENT,WORK_PROVINCE,EDU_LEVEL,SALAR，HAS_FUND 出现缺失，同时，将表格中的字符串替换成相应的数字，如 IS_LOCAL 属性中本地籍设为 1，非本地籍设为 0，对于学历， 其他 0 初中 1 高中 2 专科及以下 3 专科 4 本科 5 硕士研究生 6 博士研究生 7 硕士及以上 8，等等如此处理。

（二）数据可视化

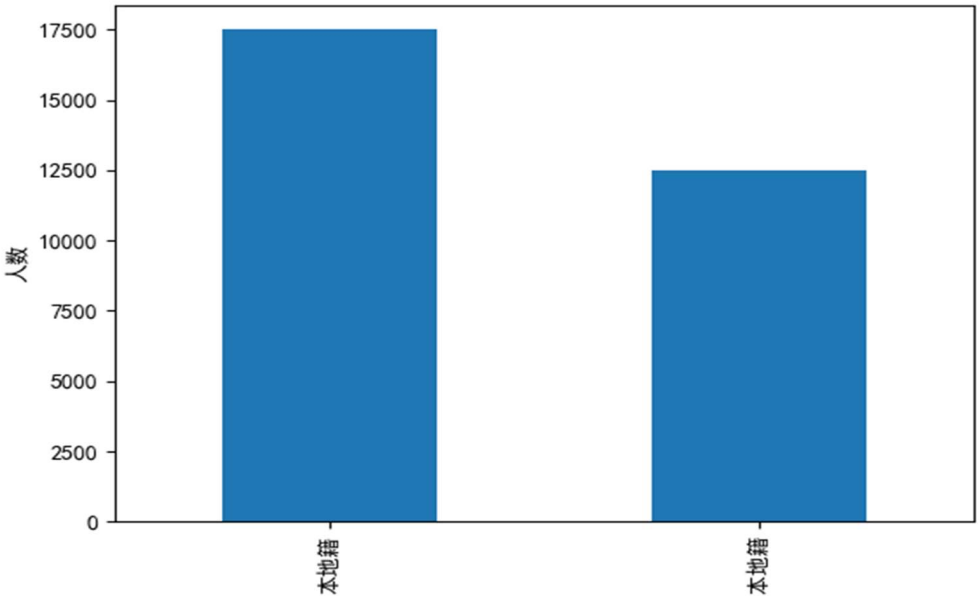
用 matplotlib 画出训练集各属性与 Y 值之间的关系，用于初步数据分类。未逾期客户人数与逾期客户人数情况如图 1 所示：

图 1 未逾期客户人数与逾期客户人数



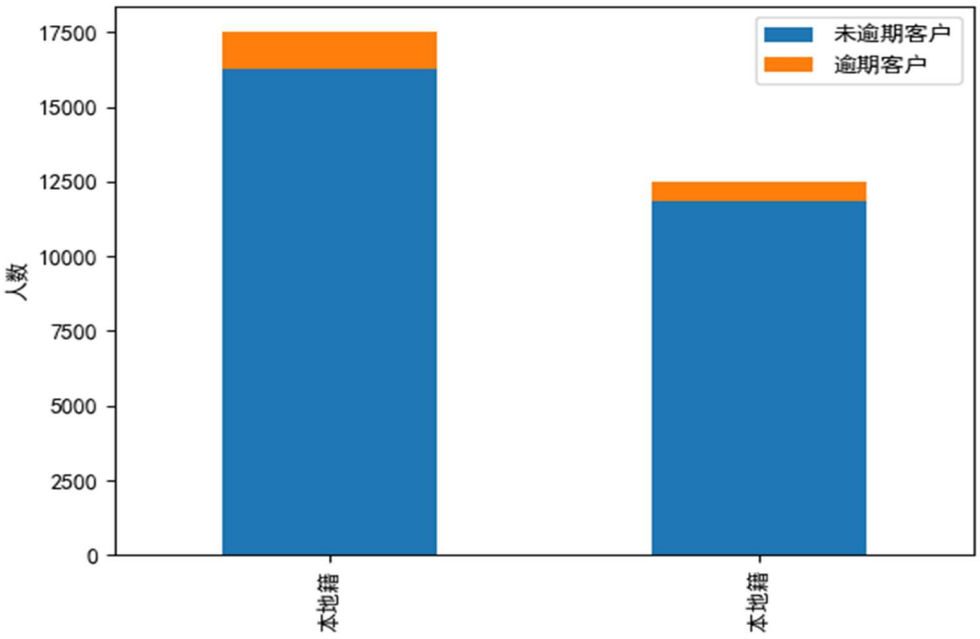
在逾期客户中，本地籍和非本地籍人数情况如图 2 所示：

图 2 逾期客户中本地籍与非本地籍人数



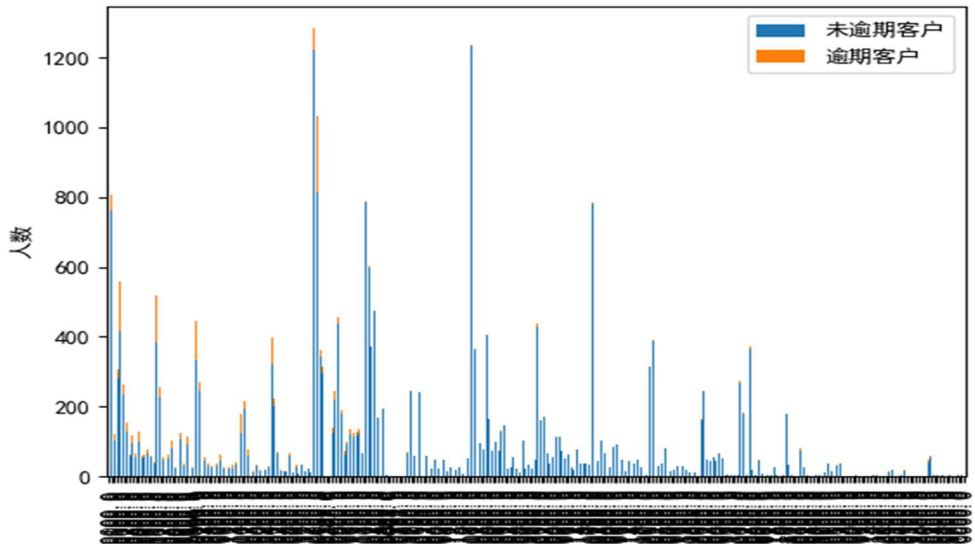
为观察户籍因素对客户是否逾期的影响，作图 3：

图 3 户籍因素对逾期情况的影响



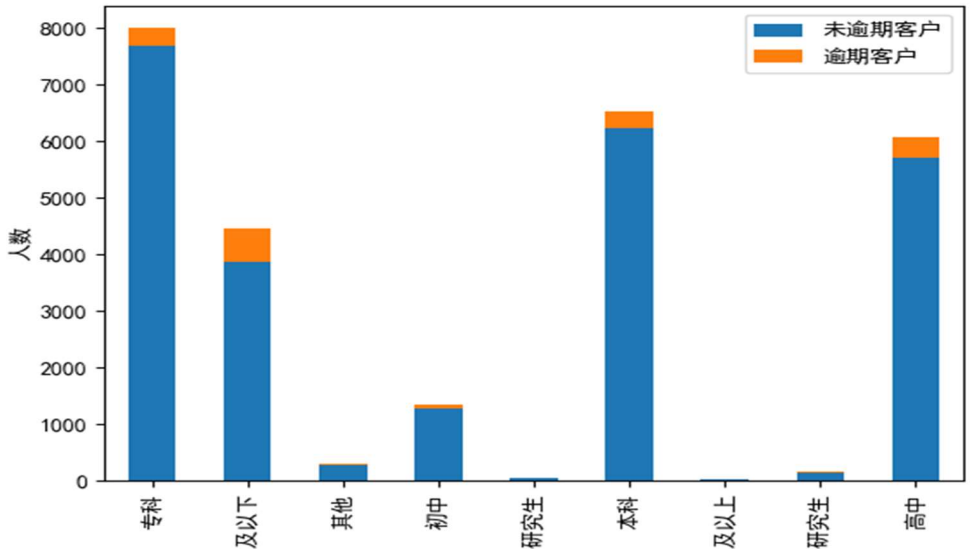
工作省份对客户是否逾期的情况如图 4 所示：

图 4 工作省份对逾期情况的影响



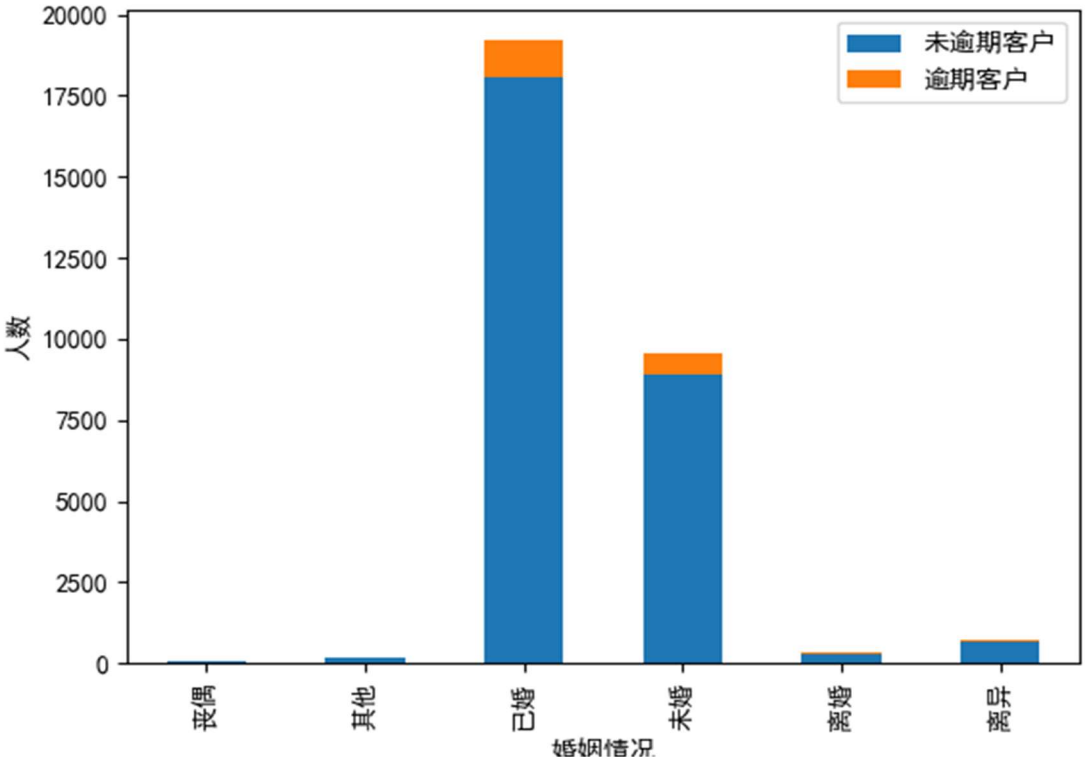
学历对客户是否逾期的影响情况如图 5 所示：

图 5 学历与逾期情况分析



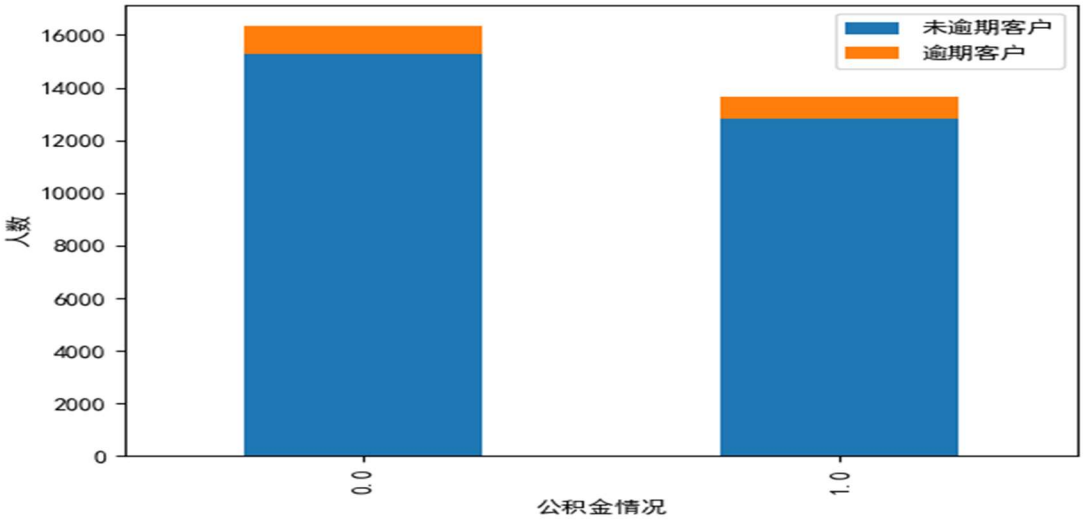
婚姻状况对是否逾期的影响情况如图 6 所示：

图 6 婚姻状况对逾期的影响



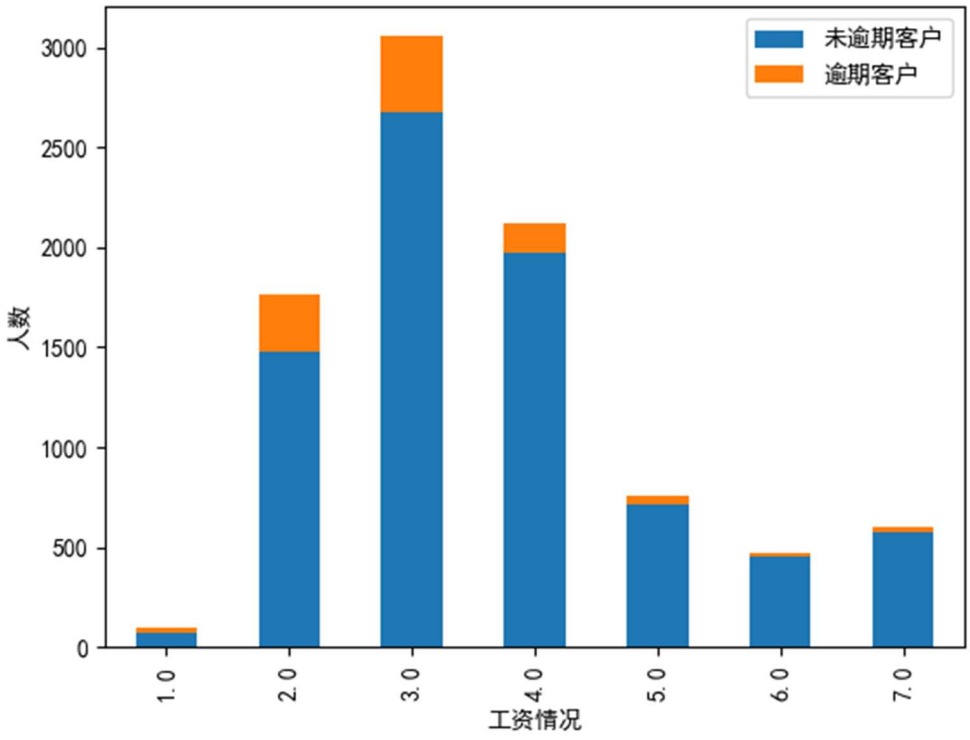
公积金对逾期情况的影响如图 7 所示：

图 7 公积金对逾期情况的影响



工资对逾期情况的影响如图 8 所示：

图 8 工资对逾期情况的影响



（三）缺失值处理

随机森林算法（Random forest algorithm）是对 bagging 算法的扩展。除了仍然根据从训练数据样本建立复合模型之外，随机森林对用做构建树（tree）的数据特征做了一定限制，使得生成的决策树之间没有关联，从而提升算法效果。

随机森林算法

决策树运行的每一步都涉及到对数据集中的最优分裂点（best split point）进行贪婪选择（greedy selection）。

这个机制使得决策树在没有被剪枝的情况下易产生较高的方差。整合通过提取训练数据库中不同样本（某一问题的不同表现形式）构建的复合树及其生成的预测值能够稳定并降低这样的高方差。这种方法被称作引导聚集算法（bootstrap aggregating），其简称 bagging 正好是装进口袋，袋子的意思，所以被称为「装袋算法」。该算法的局限在于，由于生成每一棵树的贪婪算法是相同的，那么有可能造成每棵树选取的分裂点（split point）相同或者极其相似，最终导致不同树之间的趋同（树与树相关联）。相应地，反过来说，这也使得其会产生相似的预测值，降低原本要求的方差。

我们可以采用限制特征的方法来创建不一样的决策树，使贪婪算法能够在建树的同时评估每一个分裂点。这就是随机森林算法（Random Forest algorithm）。

与装袋算法一样，随机森林算法从训练集里撷取复合样本并训练。其不同之处在于，数据在每个分裂点处完全分裂并添加到相应的那棵决策树当中，且可以只考虑用于存储属性的某一固定子集。

缺失值这种情况在现实问题中非常普遍，这会导致一些不能处理缺失值的分析方法无法应用，因此，在信用风险评级模型开发的第一步我们就要进行缺失值处理。缺失值处理的方法，包括如下几种。

（1） 直接删除含有缺失值的样本。

（2） 根据样本之间的相似性填补缺失值。

（3） 根据变量之间的相关关系填补缺失值。

在本案例中，对于 AGENT，由第二步的数据可视化发现与 Y 值关系程度很低，并且缺失值太多，考虑舍弃。参考其他表格的数据，填补出 HAS_FUND（缺失值仅有一个，故而我们选择随机填充）和 EDU_LEVEL 的数据。对于 EDU_LEVEL 的数据，我们选择分段填充，如果 EDU_LEVEL 和 SALATY 同时不缺失，则我们根据他的工资大概估计他的学历；如果都缺失，则填充众数。WORK_PRIVINCE 无法从其他表格获取，且用 LASSO 原理进行特征关联度分析时，发现 WORK-PROVINCE 与 Y

值得 coef 系数为 0，故舍弃，我们将其转换成 development_level 这一属性，这也是极为合理的操作，因为工作城市的发展水平很大程度影响到一个人的工资水平，故而我们根据全国各个省市的平均工薪水平进行等级制度划分，缺失值则填充众数。

随后对于 salary 用随机森林算法进行填补。

三、指标选择

（一）LassoCV 初步特征选择

1、LassoCV 原理

LASSO（ least absolute shrinkage and selection operator，最小绝对值收缩和选择算子）方法与岭回归和 LARS（least angle regression，最小角回归）很类似。与岭回归类似，它也是通过增加惩罚函数来判断、消除特征间的共线性。与 LARS 相似的是它也可以用作参数选择，通常得出一个相关系数的稀疏向量。

Lasso 是一种估计稀疏线性模型的方法.由于它倾向具有少量参数值的情况，对于给定解决方案是相关情况下，有效的减少了变量数量。 因此，Lasso 及其变种是压缩感知(压缩采样)的基础。在约束条件下，它可以回复一组非零精确的权重系数。用数学形式表达，Lasso 包含一个使用 ell_1 先验作为正则化因子的线性模型。其目标函数是最小化:

$$\min_{w} \frac{1}{2n_{samples}}||Xw-y||_2^2+\alpha||w||_1$$

Lasso 解决带 $\alpha||w||_1$ 惩罚项的最小平方和，其中 α 是一个常量，$||w||_1$ 是参数向量的 ℓ_1-norm。Lasso 类实现使用了坐标下降法(一种非梯度优化算法)来拟合系数。

2、合并表格并标准化

将训练集与所有其他非测试集的表格合并，去掉缺失值，用剩下的数据进行第一步的数据预处理，字符串转化成数字。

3、特征选择

采用 LassoCV 模型，以除了‘Y’之外的特征作为 X,’ Y’作为目标 y,选择出 coef 不是 0 的特征，作为第一步的粗选特征。

4、信用评分模型的变量选择方法

用上一步的得到的粗选特征，我们进行 WOE 分析方法，即是通过比较指标分箱和对应分箱的违约概率来确定指标是否符合经济意义。

5、分箱处理

变量分箱（binning）是对连续变量离散化（discretization）的一种称呼。信用评分卡开发中一般有常用的等距分段、等深分段、最优分段。其中等距分段（Equval length intervals）是指分段的区间是一致的，比如年龄以十年作为一个分段；等深分段（Equal frequency intervals）是先确定分段数量，然后令每个分段中数据数量大致相等；最优分段（Optimal Binning）又叫监督离散化（supervised discretizaion），使用递归划分（Recursive Partitioning）将连续变量分为分段，背后是一种基于条件推断查找较佳分组的算法。

我们首先选择对连续变量进行最优分段，在连续变量的分布不满足最优分段的要求时，再考虑对连续变量进行等距分段。

6、WOE

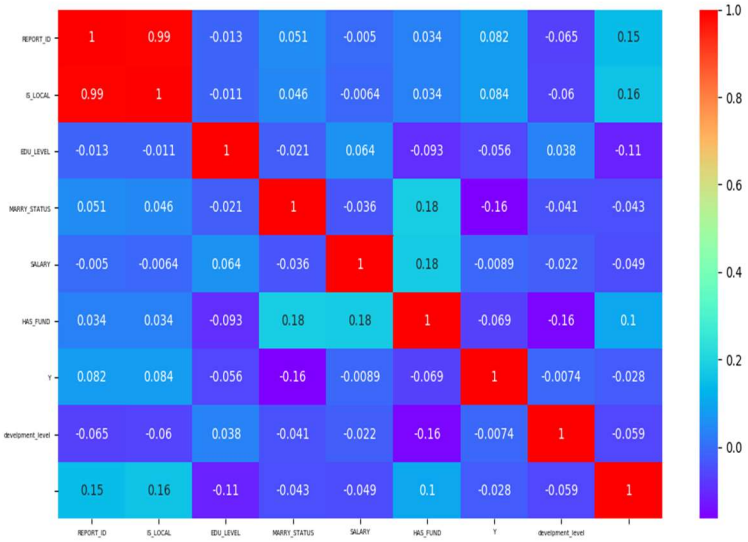
WoE 分析， 是对指标分箱、计算各个档位的 WoE 值并观察 WoE 值随指标变化的趋势。其中 WoE 的数学定义是:woe=ln(goodattribute/badattribute)在进行分析时，我们需要对各指标从小到大排列，并计算出相应分档的 WoE 值。其中正向指标越大，WoE 值越小；反向指标越大，WoE 值越大。正向指标的 WoE 值负斜率越大，反响指标的正斜率越大，则说明指标区分能力好。WoE 值趋近于直线，则意味指标判断能力较弱。若正向指标和 WoE 正相关趋势、反向指标同 WoE 出现负相关趋势，则说明此指标不符合经济意义，则应当予以去除。

WoE 函数实现在上一节的 mono_bin()函数里面已经包含，这里不再重复。

7、相关性分析和 IV 筛选

接下来，我们会用经过清洗后的数据看一下变量间的相关性。注意，这里的相关性分析只是初步的检查，进一步检查模型的 VI（证据权重）作为变量筛选的依据。

图 9 变量之间相关性图

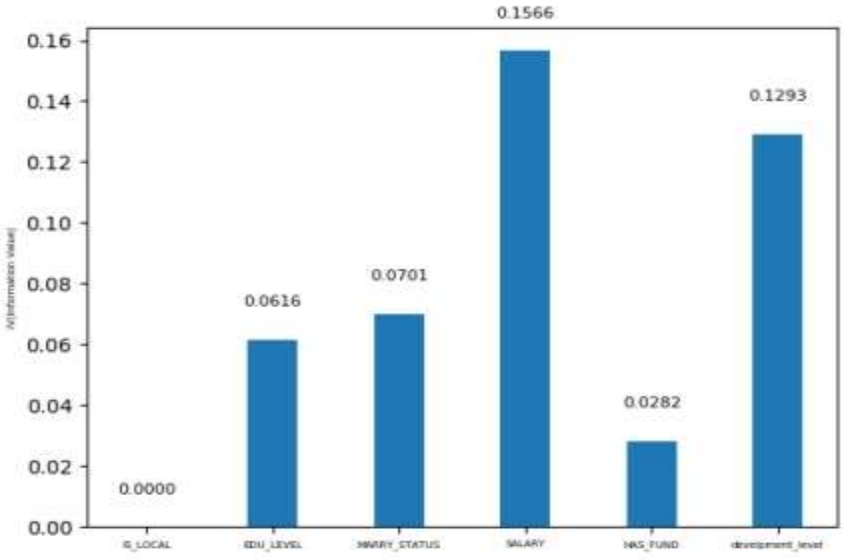


接下来，我进一步计算每个变量的 Infomation Value（IV）。IV 指标是一般用来确定自变量的预测能力。 其公式为：

$$IV = \sum (g - b) \log \frac{g}{b}$$

如图 10：

图 10



8、WOE 转换

证据权重（Weight of Evidence,WOE）转换可以将 sklearn 中的各种回归模型转变为标准评分卡格式。引入 WOE 转换的目的并不是为了提高模型质量，只是一些变量不应该被纳入模型，这或者是因为它们不能增加模型值，或者是因为与其模型相关系数有关的误差较大，其实建立标准信用评分卡也可以不采用 WOE 转换。这种情况下，sklearn 中的各种回归模型需要处理更大数量的自变量。尽管这样会增加建模程序的复杂性，但最终得到的评分卡都是一样的。

在建立模型之前，我们需要将筛选后的变量转换为 WoE 值，便于信用评分。

9、划分 train 集

由于官方发布的 test 文件没有 Y 值，预测结果无法评价，于是采用 train_test_split 方法随机切分第四步形成的 WoE 数据，将 1/4 的数据作为测试集，3/4 的数据作为训练集。

四、建立模型

采用四种模型，分别是 LogisticRegression 模型，DecisionTreeClassifier 模型，RidgeClassifier 模型，SGDClassifier 模型。并且分别用上一步得出的训练集代入模型中，训练，得出模型，保存。

LogisticRegression 原理

1、考虑有 K 种分类的场合。用 G 来作为观测到的分类结果。

2、采用对区域分划的手段来进行特征分类。

3、可以得出 Gˆ=argmaxGP(G|X)Gˆ=arg maxGP(G|X)为最优的预测结果。

4、还需要使用一些单调变换保证预测值在[0,1]之间，并得到相应的线性关系形式。因此，我们对于每个分类 k，假设

$$P(G=k|X=x)=\frac{e^{X\beta_k}}{1+\sum_{k=1}^{K-1}e^{X\beta_k}} \text{且} P(G=k|X=x)=\frac{1}{1+\sum_{k=1}^{K-1}e^{X\beta_k}}$$

5、进一步的，我们取任意类 K 作为对照组，且各组相加概率之和必为 1

进而得出

$$P(G=k|X=x)=\frac{e^{X\beta_k}}{1+\sum_{k=1}^{K-1}e^{X\beta_k}} \text{且} P(G=k|X=x)=\frac{1}{1+\sum_{k=1}^{K-1}e^{X\beta_k}}$$

6、最终得到两组之间的概率比值为：

$$\frac{P(G=k|X=x)}{P(G=K|X=x)}=e^{X\beta_k} \Rightarrow \log\left(\frac{P(G=k|X=x)}{P(G=K|X=x)}\right)=X\beta_k$$

最后求解的时候，用最大似然准则，来求解$\max L(\beta)=\max\sum_{i=1}^{N}\log P(G_i\mid X_i)$

7、下面以 0、1 分类问题，即 K=2 为例，来展示求解过程。

8、我们这个时候有两类，不妨记作 1 和 0，则

$$P(G=1\mid X=x)=\frac{e^{X\beta}}{1+e^{X\beta}}=\sigma(X\beta)$$

9、因而他的对数似然函数：

$$L(\beta)=\sum_{i=1}^{N}\left[G_i\log P(G_i=1\mid X=x_i)\right]+(1-G_i)\log P(G_i=0\mid X=x_i)$$

$$L(\beta)=\sum_{i=1}^{N}\left[G_iX_i\beta+\log\left(1-\sigma(X_i\beta)\right)\right]$$

10、然后求导：$\frac{\partial L(\beta)}{\partial\beta}=\sum_{i=1}^{N}\left[G_iX_i-\sigma(X_i\beta)X_i\right]=\sum_{i=1}^{N}\left[G_i-\sigma(X_i\beta)\right]X_i$

11、继而可以用牛顿法迭代求数值解：

$$\beta^{new}=\beta^{old}\pm(\frac{\partial^2\mathcal{L}(\beta)}{\partial\beta\partial\beta})^{-1}\frac{\partial\mathcal{L}(\beta)}{\partial\beta}$$

，其中的二阶导数部分可以化简为：

$$\frac{\partial^2\mathcal{L}(\beta)}{\partial\beta\partial\beta}=-\sum_{i=1}^{N}X_i'X_i[\sigma(X_i\beta)(1-\sigma(X_i\beta))]$$

12、记 $\Sigma(\beta)=\begin{bmatrix}\sigma(X_1\beta)\\ \vdots\\ \sigma(X_N\beta)\end{bmatrix}$，且

$$W=\begin{bmatrix}\sigma(X_1\beta)(1-\sigma(X_1\beta)) & & \\ & \ddots & \\ & & \sigma(X_N\beta)(1-\sigma(X_N\beta))\end{bmatrix}$$

13、因此，

$$\beta^{new} \leftarrow \beta^{old} + (X'WX)^{-1}X'(G - \Sigma(\beta^{old})) = \underbrace{(X'WX)^{-1}X'W}_{OLS}\underbrace{[X\beta^{old} + W^{-1}(G - \Sigma(\beta^{old}))]}_{\tilde{z}}$$

14.经过一系列的简化，这里相当于加权的最小二乘法，目标函数为

$$\arg\min_{\beta^{new}}(Z - X\beta)'W(Z - X\beta)$$

接下来，使用梯度下降方法（SDG）进行回归。

SGDClassifier 本质上是一个线性核函数的模型，它使用了随机下降做训练，这样的优势在我们并不是每次都使用了全部的样本，收敛速度会快很多。SGDClassifier 每次只使用一部分做训练，在这种情况下，每次只会拿下一个部分待训练在本次做评估，然后训练完之后，再在这个部分上做一次评估，看看是否有优化。也就是说， 随机梯度下降是通过每个样本来迭代更新一次，如果样本量很大的情况（例如几十万），那么可能只用其中几万条或者几千条的样本，就已经迭代到最优解了。

DecisionTreeClassifier 原理

1、构建单棵回归树（regression tree）

考虑数据集 D={(xi,yi),1≤i≤N}D={(xi,yi),1≤i≤N}。对不同的节点根据不同的门限来分类，并逐层分叉。

2、对于一颗树 T，我们采用如下记号：

|T||T|:叶子的总数

Rm ：1≤m≤|T|，某个叶子或者根节点。

Nm：叶子节点 Rm 中的样本数。

Cm：$\frac{1}{Nm}$ Σyi，这 Nm 个点 y 的平均值。

$\sigma_m^2:\frac{1}{Nm}\sum(y_4\text{-}C_m)^2$，每个 Rm 中的均方误差（方差）。

一棵树的质量 Q(T)定义为 Q(T)=$\sum_{m=1}^{M}\delta_m^2 N_m$。给定一棵树，有一个函数 y=f(x|T)y=f(x|T)，然后就可以预测了。

3、选择叶子和层次以控制树的生长，我们一共有 P(n+1)种选择。需要从中选出最好的 Pi 和 ti。当树生长不动的时候，停止分支。若分支过细导致过拟合时，我们需要剪枝。

引入 Q(T)+α |T|作为惩罚项，并使用 cross-validation 或者 bootstrap 算法进行剪枝。

4、 分类树

重新定义新的准则，类似于 0-1 准则。$\hat{P}_{km}=\frac{1}{Nm}\sum 1(y_4=k)$ 也就是节点中属于第 k 类的比例，所以$\sum_k P_k=1$。

这样我们就有$\hat{P}_{km}=\max_k P_{km}$，即主导类别占据该节点。

采用基尼准则（Gini）作为预测标准，定义函数 Gm=$\sum_k \hat{P}_{km}(1-P_{km})$ 然后 Q(T)=$\sum_m N_m G_m$

5、 决策树预测

设定循环次数 B.通过 bootstrap 方法生成一个自生样本 Db。随机选取 m(≤p≤p)个变量（相应的$D_b^p \to D_b^m$，取了 m 维子集）。由D_b^m 生成树 TbTb。输出{$T_b(x)$}$_{b=1}^{B}$(即森林）。

在预测时取均值，最终输出$\hat{f}(x)=\frac{1}{B}\sum_{b=1}^{B}T_b(x)$。分类分类的情况下进行投票，$\hat{G}(x)=\max\ \ vote(\{T_b(x)_{b=1}^{B}\})$ 得票最多的那类获胜。

SGDClassifier

这种方法本质上是一个线性核函数的模型，它使用了随机下降做训练，这样的优势在我们并不是每次都使用了全部的样本，收敛速度会快很多。SGDClassifier 每次只使用一部分做训练，在这种情况下，每次只会拿下一个部分待训练在本次做评估，然后训练完之后，再在这个部分上做一次评估，看看是否有优化。也就是说， 随机梯度下降是通过每个样本来迭代更新一次，如果样本量很大的情况（例如几十万），那么可能只用其中几万条或者几千条的样本，就已经迭代到最优解了。

五、模型融合

用 bagging 方法。Bagging 是并行式集成学习代表方法。基于“自助采样法”（bootstrap sampling）。自助采样法机制：给定包含 m 个样本的数据集，我们先随机取出一个样本放入采样集中，再把该样本放回初始数据集，使得下一次采样时该样本还会被采到。这样，经过 m 次样本采集，我们得到包含 m 个样本的采样集。采样集中，有的样本出现过很多次，有的没有出现过。Bagging 机制：我们采样出 T 个含 m 个样本的采样集。然后基于每个采样集训练出一个学习器，再将学习器进行结合。对分类任务使用投票法，对回归任务采用平均值法。

六、交叉验证

用第六步划分出的模型，预测第五步得出的测试集，得出准确率如下：

表 3

模型	逻辑回归	决策树	Ridge	SGD
准确率	93.80%	96．87%	93.80%	93.80%

得出决策树模型的准确率最高，即是在此四类中的最佳模型。所以我们采用决策树模型预测逾期的概率 P。

七、评分模型

（一）信用评分转换公式

为了能更好地用于实际工作中，接下来，需要将预测的违约概率转化为对客户

的信用评分。转化方法为：

$$Score = A - B \times \ln(\frac{p}{1-p})$$

其中，Score 为客户的信用评分，A、B 为常数，p 为预测的违约概率。

为了得到 A、B 具体的值，做出如下设定：当 $\frac{p}{1-p}$ 为 $\frac{1}{30}$ 时，得分为 650；$\frac{p}{1-p}$ 为 $\frac{1}{60}$ 时，得分为 680；由此计算可得，A=502.79，B=43.28。

参数 A、B 的值确定后，就可以据此列出违约概率对应的评分，如下所示：

分值	$\frac{p}{1-p}$	预测违约概率 P(%)
740	1/240	0.41
710	1/120	0.83
680	1/60	1.64
650	1/30	3.23
620	1/15	6.25
590	1/7.5	11.76
560	1/3.75	21.05
530	1/1.875	37.48
500	1/0.9875	51.61

（二）信用评分构建

我们将在前面通过 logsitic 回归得到的模型的宏程序%credictScore，对前面分出来的含有 10000 个观测值的验证集做出预测，得到其违约概率，再将测试集数据计算出的预测概率代入转化公式，得到相应的信用评分，在 SAS 中使用交互式数据分析模块作图如图 11 和图 12 所示：

图 11

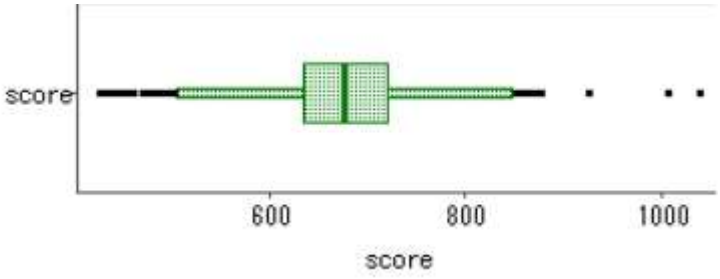
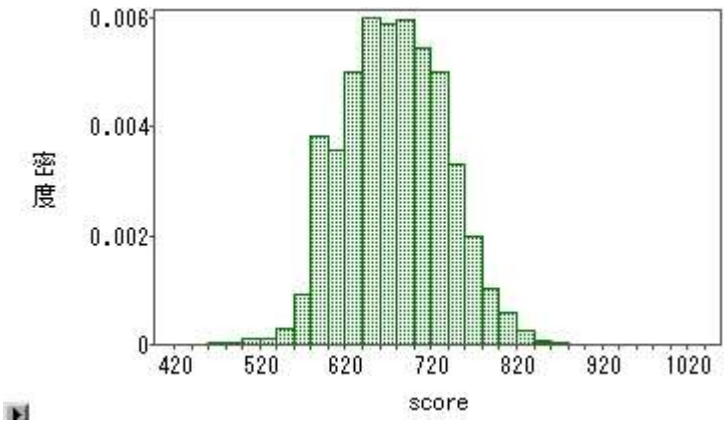


图 12



从图中可见，大部分的客户得分集中在 600-800 分段，且得分分布近似符合正态分布，说明评分结果符合人们的一般观点，这说明本评分模型具有一定的合理性与可操作性。

八、结论和建议

我们用最终用决策树模型对客户的进行了合理的信用评分。但潜在的问题是我们对表格数据采取的仅仅是表面的处理，有些变量可能十分重要，但并没有显

性的体现在具体的数据上，或者说模型本身的属性掩盖了该变量应有的影响力。所有最终预测的准确性可能会收到影响，虽然我们采取了多种模拟方法来尽量避免这一问题，但如果有更有经验的专业人员参与，可能会对变量有更深层次的处理，也可以对客户的信息有更准确的把握和分析，这样我们就可以最大程度的利用该客户的所有信息，从而做出更为准确的判断。

参考目录

[1] 盛洁. 商业银行信用卡违约概率评估的实证研究[D]. 厦门大学, 2014.

[2] 马姆杜·雷法特. 信用风险评分卡研究:基于 SAS 的开发与实施[M]. 社会科学文献出版社, 2013.

[3] 田苗苗. 数据挖掘之决策树方法概述[J]. 长春大学学报, 2004, 14(6):48-51.

附录（数据分析程序）

（一）数据预处理部分

```
import pandas as pd
from keras import Sequential
from keras.layers.core import Dense,Activation,Dropout
from sklearn.model_selection import train_test_split
# from matplotlib.pyplot import plt
train_data=pd.read_csv('D:\sufe\A\contest_basic_train.tsv',sep='\t')
train_data=train_data.drop(['REPORT_ID',''ID_CARD'','LOAN_DATE'],1)
train_data=train_data.dropna()
# print(train_data.info())
X=train_data.drop(['Y'],1).as_matrix()#7
y=train_data['Y'].as_matrix()#1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

model=Sequential()
model.add(Dense(14,input_shape=(7,)))
model.add(Activation('relu'))
model.add(Dense(1))
model.add((Dropout(0.3)))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()

model.fit(X_train,y_train,epochs=10000,batch_size=16)
t=model.predict(X_test)

rate=0

for i in range(len(t)):
    if t[i]==y_test[i]:
        rate+=1
```

```
        else:
            pass
    rate=1.0*rate/len(t)

    print(rate)



    # test_data=pd.read_csv('D:\sufe\A\contest_basic_test.tsv',sep='\t')
    # test_data=test_data.dropna()
    # test_data=test_data.drop(['REPORT_ID',"ID_CARD",'LOAN_DATE'],1)
```

（二）数据可视化部分

```
    def DeleteDuplicatedElementFromList(list):
        resultList = []
        for item in list:
            if not item in resultList and str(item)!="nan":
                resultList.append(item)
        return resultList
    import pandas as pd
    #coding:utf-8
    import matplotlib.pyplot as plt
    import numpy
    plt.rcParams['font.sans-serif']=['SimHei']
    plt.rcParams['axes.unicode_minus'] =False
    import numpy as np
    from pandas import Series,DataFrame
    data_train=pd.read_csv("D:\sufe\A\contest_basic_train.tsv", sep='\t')

    print(data_train.info())
    print(data_train.describe())

    d1=pd.DataFrame(columns=['特征','未逾期比例','逾期比例'])
```

```
fig=plt.figure("Y 值分类统计图")
fig.set(alpha=0.2)
data_train.Y.value_counts().plot(kind='bar')#我用柱状图
# plt.title(u"Y=0 代表未逾期客户，Y=1 代表逾期客户")
plt.ylabel(u"人数")
plt.savefig("Y 值分类统计图.png")
fig=plt.figure('户籍统计情况图')
data_train.IS_LOCAL.value_counts().plot(kind='bar')
# plt.title(u"户籍情况")
plt.ylabel(u"人数")
plt.savefig("户籍统计情况图.png")

# 属性对于目标变量的关联性统计（IS_LOCAL AGENT WORK_PROVINCE,EDU_LEVEL,MARRY_STATUS,SALATY,HAS_FUND)


#IS_LOCAL

Y1=data_train.IS_LOCAL[data_train.Y==0].value_counts()
Y2=data_train.IS_LOCAL[data_train.Y==1].value_counts()
df=pd.DataFrame({u'未逾期客户':Y1,u'逾期客户':Y2})
df.plot(kind='bar',stacked=True)
# plt.title(u"户籍因素中是否是逾期客户分析")
plt.xlabel(u"是否本地户籍")
plt.ylabel(u"人数")
print("本地与非本地因素中客户逾期分析")
print("未逾期客户人数\n",Y1)
print("逾期客户人数\n",Y2)
print("本地户籍中未逾期比例是",1.0*Y1["本地籍"]/(Y1["本地籍"]+Y2["本地籍"]),"  逾期比例是："
      ,1.0*Y2["本地籍"]/(Y1["本地籍"]+Y2["本地籍"]))
```

```
print("非本地户籍中逾期比例是",1.0*Y1["非本地籍"]/(Y1["非本地籍"]+Y2["非本地籍"]),"
逾期比例是："
      ,1.0*Y2["非本地籍"]/(Y1["非本地籍"]+Y2["非本地籍"]))
d1.iat["本地籍",0]=1.0*Y1["本地籍"]/(Y1["本地籍"]+Y2["本地籍"])
d1.iat["本地籍",1]=1.0*Y2["本地籍"]/(Y1["本地籍"]+Y2["本地籍"])
d1.iat["非本地籍",0]=1.0*Y1["非本地籍"]/(Y1["非本地籍"]+Y2["非本地籍"])
d1.iat["非本地籍",1]=1.0*Y2["非本地籍"]/(Y1["非本地籍"]+Y2["非本地籍"])
plt.savefig('户籍对于逾期的影响.png')
#WORK_PRO 工作省份
Y3=data_train.WORK_PROVINCE[data_train.Y==0].value_counts()
Y4=data_train.WORK_PROVINCE[data_train.Y==1].value_counts()
df=pd.DataFrame({u'未逾期客户':Y3,u'逾期客户':Y4})
df.plot(kind='bar',stacked=True)
# plt.title(u"工作省份中是否是逾期客户分析")
plt.xlabel(u"工作省份")
plt.ylabel(u"人数")
plt.savefig('工作省份对于逾期的影响.png')
# listA=data_train["WORK_PROVINCE"]
# b=DeleteDuplicatedElementFromList(listA)
#
# print(b)
# for i in b:
#       print(i," 中 的 未 逾 期 比 例 ",1.0*Y3[i]/(Y3[i]+Y4[i])," 逾 期 比 例 是
",1.0*Y4[i]/(Y3[i]+Y4[i]))

#EDU_LEVEL
Y3=data_train.EDU_LEVEL[data_train.Y==0].value_counts()
Y4=data_train.EDU_LEVEL[data_train.Y==1].value_counts()
df=pd.DataFrame({u'未逾期客户':Y3,u'逾期客户':Y4})
df.plot(kind='bar',stacked=True)
# plt.title(u"学历与逾期客户分析")
plt.xlabel(u"学历")
```

```
plt.ylabel(u"人数")
print("学历与是否是逾期客户的分析")
print("未逾期客户人数：\n",Y3)
print("逾期客户人数：\n",Y4)
d2=pd.DataFrame(columns=['未逾期比例','逾期比例'])
b=["专科","本科","高中","专科及以下","初中","其他","硕士研究生","博士研究生","硕士及以上"]
d2.index=b
for i in b:
    print(i," 中 的 未 逾 期 比 例 ",1.0*Y3[i]/(Y3[i]+Y4[i])," 逾 期 比 例 是 ",1.0*Y4[i]/(Y3[i]+Y4[i]))
    d2.iat[i,0]=1.0*Y3[i]/(Y3[i]+Y4[i])
    d2.iat[i,1]=1.0*Y4[i]/(Y3[i]+Y4[i])
plt.savefig('学历的影响.png')
#Marry_STATUS
Y3=data_train.MARRY_STATUS[data_train.Y==0].value_counts()
Y4=data_train.MARRY_STATUS[data_train.Y==1].value_counts()
df=pd.DataFrame({u'未逾期客户':Y3,u'逾期客户':Y4})
df.plot(kind='bar',stacked=True)
# plt.title(u"婚姻情况与逾期客户分析")
plt.xlabel(u"婚姻情况")
plt.ylabel(u"人数")
d3=pd.DataFrame(columns=['未逾期比例','逾期比例'])
print("婚姻情况与逾期情况分析")
print("未逾期客户人数：\n",Y3)
print("逾期客户人数：\n",Y4)
b=["丧偶","其他","已婚","未婚","离婚","离异"]
d3.index=b
for i in b:
    print(i," 中 的 未 逾 期 比 例 ",1.0*Y3[i]/(Y3[i]+Y4[i])," 逾 期 比 例 是 ",1.0*Y4[i]/(Y3[i]+Y4[i]))
    d3.iat[i,0]=1.0*Y3[i]/(Y3[i]+Y4[i])
    d3.iat[i,1]=1.0*Y4[i]/(Y3[i]+Y4[i])
```

```
plt.savefig('婚姻的影响.png')
#has_fund
Y3=data_train.HAS_FUND[data_train.Y==0].value_counts()
Y4=data_train.HAS_FUND[data_train.Y==1].value_counts()
df=pd.DataFrame({u'未逾期客户':Y3,u'逾期客户':Y4})
df.plot(kind='bar',stacked=True)
# plt.title(u"公积金与逾期客户分析")
plt.xlabel(u"公积金情况")
plt.ylabel(u"人数")
print("公积金（0-无公积金,1-有公积金）与逾期客户分析")
b=[0,1]
d4=pd.DataFrame(columns=['未逾期比例','逾期比例'])
d4.index=b
for i in b:
    print(i," 中 的 未 逾 期 比 例 ",1.0*Y3[i]/(Y3[i]+Y4[i])," 逾 期 比 例 是 ",1.0*Y4[i]/(Y3[i]+Y4[i]))
    d4.iat[i,0]=1.0*Y3[i]/(Y3[i]+Y4[i])
    d4.iat[i,1]=1.0*Y4[i]/(Y3[i]+Y4[i])
plt.savefig('公积金的影响.png')
#salary
print("salary 的基本情况")
print(data_train["SALARY"].describe())
Y3=data_train.SALARY[data_train.Y==0].value_counts()
Y4=data_train.SALARY[data_train.Y==1].value_counts()
df=pd.DataFrame({u'未逾期客户':Y3,u'逾期客户':Y4})
df.plot(kind='bar',stacked=True)
# plt.title(u"工资与逾期客户分析")
plt.xlabel(u"工资情况")
plt.ylabel(u"人数")
plt.savefig('工资的影响.png')
print("有工资记录下的逾期比例分析")
b=[1,2,3,4,5,6,7]
```

```
    d5=pd.DataFrame(columns=['未逾期比例','逾期比例'])
    d5.index=b
    for i in b:
        print(i,"中的未逾期比例",1.0*Y3[i]/(Y3[i]+Y4[i]),"    逾期比例是",1.0*Y4[i]/(Y3[i]+Y4[i]))
        d5.iat[i,0]=1.0*Y3[i]/(Y3[i]+Y4[i])
        d5.iat[i,1]=1.0*Y4[i]/(Y3[i]+Y4[i])
    # print("无工资记录情况下的比例分析")
    # nanSalary=data_train.groupby(by=["SALARY","Y"])
    # print(nanSalary.size())
    # num0=0
    # num1=0
    # numn=0
    # for i in range(len(data_train)):
    #     if str(data_train['SALARY'])=="nan" and data_train['Y']==0:
    #         num0=num0+1
    #         numn=numn+1
    #     elif str(data_train['SALARY'])=="nan" and data_train['Y']==1:
    #         num1=num1+1
    #         numn=numn+1
    # print(1.0*num0/numn,1.0*num1/numn)
    d1=d1.append(d2,ignore_index=False)
    print(d1)
    plt.show()
```

（三）Adaptive-Lasso 变量选择算法

```
    from sklearn.linear_model import LassoCV
    import pandas as pd

    train_data=pd.read_csv('D:\sufe\A\data_train_changed.csv')
    train_data=train_data.ix[0:,1:].drop(['REPORT_ID',"ID_CARD",'LOAN_DATE'],1)
    train_data=train_data.dropna()
```

```
    # print(train_data.info())
    X=train_data.drop(['Y'],1).as_matrix()#7
    y=train_data['Y'].as_matrix()#1
    lassocv = LassoCV()
    lassocv.fit(X,y)
    print(train_data.columns.drop('Y'),lassocv.coef_)
```

（四）特征工程

```
#试着补全 AGENT WORK_PROVINCE EDU_LEVEL SALARY
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
#先要进行文字等级换成数字等级
#对于 IS_LOCAL 非本地籍是 0 本地籍是 1
#对于学历 其他 0 初中 1 高中 2 专科及以下 3 专科 4 本科 5 硕士研究生 6 博士研究生 7 硕士及以上 8
#对于婚姻 其他 0 丧偶 1 离婚 2 离异 3 未婚 4 已婚 5
#对于 agent weijinhui 0 orgloan 1 bestpay 2 kuaiqian 3 ifpp 4 fenqile 5 huifusdb 6 rongzhijia 7 chinapnr 8 wechat 9 APP 10
def str2level(data):
    for j in range(3,len(data.columns)):#列
        for i in range(len(data.index)):#行
            if j==100:#agent
                if data.iat[i,j]=="weijinhui":
                    data.iat[i,j]=0
                elif data.iat[i,j]=="orgloan":
                    data.iat[i, j]=1
                elif data.iat[i,j]=="bestpay":
                    data.iat[i,j]=2
                elif data.iat[i,j]=="kuaiqian":
                    data.iat[i,j]=3
                elif data.iat[i,j]=="ifpp":
                    data.iat[i, j] = 4
                elif data.iat[i,j]=="fenqile":
                    data.iat[i, j] = 5
                elif data.iat[i,j]=="huifusdb":
                    data.iat[i,j]=6
                elif data.iat[i,j]=="rongzhijia":
                    data.iat[i,j]=7
                elif data.iat[i,j]=="chinapnr":
                    data.iat[i,j]=8
                elif data.iat[i,j]=="wechat":
```

```
                data.iat[i,j]=9
            elif data.iat[i,j]=="APP":
                data.iat[i,j]=10

    if j==3:
        if data.iat[i,j]=="本地籍":
            data.iat[i, j] =1
        else:
            data.iat[i,j]=0
    if j==4:#edu_level
        if data.iat[i,j]=="其他":
            data.iat[i,j]=0
        elif data.iat[i,j]=="初中":
            data.iat[i,j]=1
        elif data.iat[i,j]=="高中":
            data.iat[i,j]=2
        elif data.iat[i,j]=="专科及以下":
            data.iat[i,j]=3
        elif data.iat[i,j]=="专科":
            data.iat[i,j]=4
        elif data.iat[i,j]=="本科":
            data.iat[i,j]=5
        elif data.iat[i,j]=="硕士研究生":
            data.iat[i,j]=6
        elif data.iat[i,j]=="博士研究生":
            data.iat[i,j]=7
        elif data.iat[i,j]=="硕士及以上":
            data.iat[i,j]=8
        # else:
        #         data.iat[i,j]=9
    if j==5:#marry_status
        if data.iat[i,j]=="其他":
            data.iat[i, j]=0
        elif data.iat[i,j]=="丧偶":
            data.iat[i, j]=1
        elif data.iat[i,j]=="离婚":
            data.iat[i, j]=2
        elif data.iat[i,j]=="离异":
            data.iat[i, j]=3
        elif data.iat[i,j]=="未婚":
            data.iat[i, j]=4
        elif data.iat[i,j]=="已婚":
            data.iat[i, j]=5
        else:
```

```
                    data.iat[i, j]=6
    return data

#先补全 fund ， 再补全 edu_level，再补全 salary


def set_missing_HASFUND(data):
    funddata=data[["HAS_FUND","IS_LOCAL","MARRY_STATUS"]]
    known_fund=funddata[funddata.HAS_FUND.notnull()].as_matrix()
    unknow_fund=funddata[funddata.HAS_FUND.isnull()].as_matrix()
    y=known_fund[:,0]
    X=known_fund[:,1:]
    rfr = RandomForestRegressor(random_state=0, n_estimators=200, n_jobs=-1)
    rfr.fit(X, y)
    predictfund = rfr.predict(unknow_fund[:, 1::])
    data.loc[(data.HAS_FUND.isnull()), 'HAS_FUND'] = predictfund
    return data



def set_missing_edu(data):
    edudata=data[["EDU_LEVEL","IS_LOCAL","MARRY_STATUS","HAS_FUND"]]
    known_edu=edudata[edudata.EDU_LEVEL.notnull()].as_matrix()
    unknown_edu=edudata[edudata.EDU_LEVEL.isnull()].as_matrix()
    y=known_edu[:,0]
    X=known_edu[:,1:]
    rfr = RandomForestRegressor(random_state=0, n_estimators=200, n_jobs=-1)
    rfr.fit(X, y)
    predictedu = rfr.predict(unknown_edu[:, 1::])
    data.loc[(data.EDU_LEVEL.isnull()), 'EDU_LEVEL'] = predictedu
    return data
def set_missing_province(data):

province=data[["WORK_PROVINCE","IS_LOCAL","MARRY_STATUS","HAS_FUND","EDU_LEVEL
"]]
    known_province=province[province.WORK_PROVINCE.notnull()].as_matrix()
    unknown_province=province[province.WORK_PROVINCE.isnull()].as_matrix()
    y = known_province[:, 0]
    X = known_province[:, 1:]
    rfr = RandomForestRegressor(random_state=0, n_estimators=200, n_jobs=-1)
    rfr.fit(X, y)
    predictpro = rfr.predict(unknown_province[:, 1::])
    data.loc[(data.WORK_PROVINCE.isnull()), 'WORK_PROVINCE'] = predictpro
    return data
def set_missing_salary(data):
```

```
salary_data=data[["SALARY","IS_LOCAL","EDU_LEVEL","MARRY_STATUS","HAS_FUND","devel
pment_level"]]
    known_salary=salary_data[salary_data.SALARY.notnull()].as_matrix()
    unknow_salary=salary_data[salary_data.SALARY.isnull()].as_matrix()
    #工资
    y=known_salary[:,0]
    #其他特征值
    X=known_salary[:,1:]
    rfr = RandomForestRegressor(random_state=0, n_estimators=200, n_jobs=-1)
    rfr.fit(X,y)
    predictsalary=rfr.predict(unknow_salary[:,1::])
    data.loc[(data.SALARY.isnull()),'SALARY']=predictsalary
    return data
def set_missing_salary_y(data):
    salary_data = data[["SALARY", "IS_LOCAL", "EDU_LEVEL", "MARRY_STATUS",
"HAS_FUND", "develpment_level","Y"]]
    known_salary = salary_data[salary_data.SALARY.notnull()].as_matrix()
    unknow_salary = salary_data[salary_data.SALARY.isnull()].as_matrix()
    # 工资
    y = known_salary[:, 0]
    # 其他特征值
    X = known_salary[:, 1:]
    rfr = RandomForestRegressor(random_state=0, n_estimators=200, n_jobs=-1)
    rfr.fit(X, y)
    predictsalary = rfr.predict(unknow_salary[:, 1::])
    data.loc[(data.SALARY.isnull()), 'SALARY'] = predictsalary
    return data
data_train=pd.read_excel("D:\sufe\dataset1.xls")
data_train=str2level(data_train)
data_train2=data_train
# print(data_train.info(),'\n',data_train2.info())
#
print(data_train.IS_LOCAL.value_counts(),'\n',data_train.MARRY_STATUS.value_counts(),'\n',d
ata_train.HAS_FUND.value_counts())
# data_train=set_missing_HASFUND(data_train)
# data_train=set_missing_edu(data_train)
# data_train=set_missing_province(data_train)
# data_train_withY=set_missing_salary_y(data_train2)
# data_train_withY.to_csv("D:\sufe\A\dataset_train_withY.csv")

data_train_withoutY=set_missing_salary(data_train)
data_train_withoutY.to_csv("D:\sufe\A\dataset_train_withoutY.csv")
```

（五）特征选择

```
import numpy as np
import pandas as pd
import scipy.stats.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
def mono_bin(Y, X, n = 20):
    r = 0
    good=Y.sum()
    bad=Y.count()-good
    while np.abs(r) < 1:
        d1 = pd.DataFrame({"X": X, "Y": Y, "Bucket": pd.qcut(X.rank(method='first'), n)})

        d2 = d1.groupby('Bucket', as_index = True)
        r, p = stats.spearmanr(d2.mean().X, d2.mean().Y)
        n = n - 1
    d3 = pd.DataFrame(d2.X.min(), columns = ['min'])
    d3['min']=d2.min().X
    d3['max'] = d2.max().X
    d3['sum'] = d2.sum().Y
    d3['total'] = d2.count().Y
    d3['rate'] = d2.mean().Y
    d3['woe']=np.log((d3['rate']/(1-d3['rate']))/(good/bad))
    d3['goodattribute'] = d3['sum'] / good
    d3['badattribute'] = (d3['total'] - d3['sum']) / bad
    iv = ((d3['goodattribute'] - d3['badattribute']) * d3['woe']).sum()
    d4 = (d3.sort_values(by='min')).reset_index(drop=True)
    print("=" * 60)
    print(d4)
    cut = []
    cut.append(float('-inf'))
    for i in range(1, n + 1):
        qua = X.quantile(i / (n + 1))
        cut.append(round(qua, 4))
    cut.append(float('inf'))
    woe = list(d4['woe'].round(3))
    return iv,cut,woe

def replace_woe(series,cut,woe):
    list=[]
    i=0
    while i<len(series):
        value=series[i]
        j=len(cut)-2
        m=len(cut)-2
```

```
            while j>=0:
                if value>=cut[j]:
                    j=-1
                else:
                    j -=1
                    m -= 1
            list.append(woe[m])
            i += 1
        return list

train_data=pd.read_csv("D:\sufe\A\dataset_train_withY.csv")
v2,cut2,woe2=mono_bin(train_data.Y,train_data.EDU_LEVEL)
v1,cut1,woe1=mono_bin(train_data.Y,train_data.IS_LOCAL)
v3,cut3,woe3=mono_bin(train_data.Y,train_data.MARRY_STATUS)
v4,cut4,woe4=mono_bin(train_data.Y,train_data.SALARY)
v5,cut5,woe5=mono_bin(train_data.Y,train_data.HAS_FUND)
v6,cut6,woe6=mono_bin(train_data.Y,train_data.develpment_level)
corr=train_data.corr()
xlabels=corr.index[1:]
ylabels=corr.index[1:]
fig=plt.figure("关联性图")
ax1 = fig.add_subplot(1, 1, 1)
sns.heatmap(corr, annot=True, cmap='rainbow', ax=ax1, annot_kws={'size': 8, 'weight': 'bold',
'color': 'blue'})
ax1.set_xticklabels(xlabels, rotation=0, fontsize=5)
ax1.set_yticklabels(ylabels, rotation=0, fontsize=5)
fig=plt.figure("Information Value 值图")
ax1 = fig.add_subplot(1, 1, 1)
ivlist=[v1,v2,v3,v4,v5,v6]
index =train_data.columns[4:].drop("Y") # x 轴的标签
x = np.arange(len(index))+1
ax1=fig.add_subplot(1,1,1)
ax1.bar(x, ivlist, width=0.4)#生成柱状图
ax1.set_xticks(x)
ax1.set_xticklabels(index, rotation=0, fontsize=5)
ax1.set_ylabel('IV(Information Value)', fontsize=5)
#在柱状图上添加数字标签
for a, b in zip(x, ivlist):
    plt.text(a, b + 0.01, '%.4f' % b, ha='center', va='bottom', fontsize=8)
# train_data['IS_LOCAL']=pd.Series(replace_woe(train_data['IS_LOCAL'],cut1,woe1))
train_data['EDU_LEVEL']=pd.Series(replace_woe(train_data['EDU_LEVEL'],cut2,woe2))
train_data['MARRY_STATUS']=pd.Series(replace_woe(train_data['MARRY_STATUS'],cut3,woe3
))
# train_data['SALARY']=pd.Series(replace_woe(train_data['SALARY'],cut4,woe4))
```

```
train_data['HAS_FUND']=pd.Series(replace_woe(train_data['HAS_FUND'],cut5,woe5))
train_data['develpment_level']=pd.Series(replace_woe(train_data['develpment_level'],cut6,wo
e6))
# train_data.drop(["REPORT_ID","ID_CARD","LOAN_DATE"],axis=1)
train_data.to_csv("D:\sufe\A\WoEdata.csv",index=False)
plt.show()
```

（六）表格合并部分

```
import pandas as pd
import numpy as np
import os

path='D:\sufe\A'
files=os.listdir(path)
train_data=pd.read_csv('D:\sufe\A\data_train_changed.csv')
data1=pd.read_csv('D:\sufe\A\contest_ext_crd_cd_ln.tsv',sep='\t')
data2=pd.read_csv('D:\sufe\A\contest_ext_crd_cd_ln_spl.tsv',sep='\t')
p=pd.merge(train_data,data1,on='REPORT_ID',how='left')
p=pd.merge(p,data2,on='REPORT_ID',how='left')
print(p.info())
```

（七）建立模型，训练模型，交叉验证部分

```
import pandas as pd
# from keras import Sequential
# from keras.layers.core import Dense,Activation,Dropout
from sklearn.model_selection import train_test_split
from sklearn import linear_model,tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.externals import joblib
from sklearn.ensemble import BaggingClassifier
import pydotplus

# # from matplotlib.pyplot import plt
# from sklearn.svm import SVC
def str2level(data):
    for j in data.columns:#列
        for i in data.index:#行
            if j=='AGENT':#agent
                if data.ix[i,j]=="weijinhui":
                    data.ix[i,j]=0
                elif data.ix[i,j]=="orgloan":
                    data.ix[i, j]=1
                elif data.ix[i,j]=="bestpay":
                    data.ix[i,j]=2
```

```
        elif data.ix[i,j]=="kuaiqian":
            data.ix[i,j]=3
        elif data.ix[i,j]=="ifpp":
            data.ix[i, j] = 4
        elif data.ix[i,j]=="fenqile":
            data.ix[i, j] = 5
        elif data.ix[i,j]=="huifusdb":
            data.ix[i,j]=6
        elif data.ix[i,j]=="rongzhijia":
            data.ix[i,j]=7
        elif data.ix[i,j]=="chinapnr":
            data.ix[i,j]=8
        elif data.ix[i,j]=="wechat":
            data.ix[i,j]=9
        elif data.ix[i,j]=="APP":
            data.ix[i,j]=10

    if j=='IS_LOCAL':
        if data.ix[i,j]=="本地籍":
            data.ix[i, j] =1
        else:
            data.ix[i,j]=0
    if j=='EDU_LEVEL':#edu_level
        if data.ix[i,j]=="其他":
            data.ix[i,j]=0
        elif data.ix[i,j]=="初中":
            data.ix[i,j]=1
        elif data.ix[i,j]=="高中":
            data.ix[i,j]=2
        elif data.ix[i,j]=="专科及以下":
            data.ix[i,j]=3
        elif data.ix[i,j]=="专科":
            data.ix[i,j]=4
        elif data.ix[i,j]=="本科":
            data.ix[i,j]=5
        elif data.ix[i,j]=="硕士研究生":
            data.ix[i,j]=6
        elif data.ix[i,j]=="博士研究生":
            data.ix[i,j]=7
        elif data.ix[i,j]=="硕士及以上":
            data.ix[i,j]=8
        # else:
        #         data.ix[i,j]=9
    if j=='MARRY_STATUS':#marry_status
```

```
                    if data.ix[i,j]=="其他":
                        data.ix[i, j]=0
                    elif data.ix[i,j]=="丧偶":
                        data.ix[i, j]=1
                    elif data.ix[i,j]=="离婚":
                        data.ix[i, j]=2
                    elif data.ix[i,j]=="离异":
                        data.ix[i, j]=3
                    elif data.ix[i,j]=="未婚":
                        data.ix[i, j]=4
                    elif data.ix[i,j]=="已婚":
                        data.ix[i, j]=5
                    else:
                        data.ix[i, j]=6
    return data


train_data=pd.read_csv('D:\sufe\A\WoEdata.csv')

train_data=train_data.ix[0:,1:].drop(['REPORT_ID',"ID_CARD",'LOAN_DATE'],1)
train_data=train_data.dropna()
# print(train_data.info())
# train_data=str2level(train_data)

X=train_data.drop(['Y'],1).as_matrix()#7

y=train_data['Y'].as_matrix()#1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)


clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6)
bagging_clf=BaggingClassifier(clf,n_estimators=20,  max_samples=0.8,  max_features=1.0,
bootstrap=True, bootstrap_features=False)
bagging_clf.fit(X_train, y_train)

treemodel=DecisionTreeClassifier()
#treemodel=BaggingClassifier(treemodel,n_estimators=20,                max_samples=0.8,
max_features=1.0, bootstrap=True, bootstrap_features=False)
treemodel.fit(X_train,y_train)

randomtree=linear_model.RidgeClassifier()
randomtree=BaggingClassifier(randomtree,n_estimators=20,              max_samples=0.8,
max_features=1.0, bootstrap=True, bootstrap_features=False)
randomtree.fit(X_train,y_train)
```

```
sgd=linear_model.SGDClassifier(tol=1e-3)
sgd=BaggingClassifier(sgd,n_estimators=20, max_samples=0.8, max_features=1.0,
bootstrap=True, bootstrap_features=False)
sgd.fit(X_train,y_train)


dot_data = tree.export_graphviz(treemodel, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("决策树.pdf")

# model=Sequential()
# model.add(Dense(2*(X_train.shape[1]),input_shape=((X_train.shape[1]),)))
# model.add(Activation('relu'))
# model.add(Dense(1))
# model.add((Dropout(0.3)))
# model.compile(loss='mean_squared_error', optimizer='adam')
# model.summary()
#
# model.fit(X_train,y_train,epochs=10000,batch_size=50 )
# svmmodel=SVC()
# svmmodel.fit(X_train,y_test)

t=bagging_clf.predict(X_test)
joblib.dump(bagging_clf,'clf.model')

z=treemodel.predict(X_test)
joblib.dump(treemodel,'treemodel.model')

w=randomtree.predict(X_test)
joblib.dump(randomtree,'randomtree.model')

s=sgd.predict(X_test)
joblib.dump(sgd,'sgd.model')

# m=model.predict(X_test)
# model.save('NNmodel.h5')

rate1=0
rate2=0
rate3=0
rate4=0
# rate5=0
for i in range(len(t)):
```

```
    if t[i]==y_test[i]:
        rate1+=1
    if z[i]==y_test[i]:
        rate2+=1
    if w[i]==y_test[i]:
        rate3+=1
    if s[i]==y_test[i]:
        rate4+=1
    # if m[i]==y_test[i]:
    #     rate5+=1

rate1=1.0*rate1/len(t)
rate2=1.0*rate2/len(t)
rate3=1.0*rate3/len(t)
rate4=1.0*rate4/len(t)
print('逻辑回归的准确率是',rate1,'\n','决策树的准确率是',rate2,'\n','Ridge 分类决策准确率是
',rate3,'\n','SGD 分类器准确率',rate4,
        '\n')
```